Towards Optimization of Cyclic Production Systems

Fabrice Chauvet, Jeffrey W. Herrmann, Jean-Marie Proth

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Towards Optimization of Cyclic Production Systems

Fabrice Chauvet - Jeffrey W. Herrmann
Jean-Marie Proth

N° 3721
Juin 1999

THÈME 4

VERS L'OPTIMISATION DES SYSTEMES DE PRODUCTION CYCLIQUES

FABRICE CHAUVET[1], JEFFREY W. HERRMANN[2] and
JEAN-MARIE PROTH[3]

RESUME

Dans ce papier, l'expression "système de production" désigne des lignes de fabrication, des ateliers, des systèmes d'assemblage, des systèmes KANBAN et, plus généralement, tout système à évènements discrets (SED) qui transforme de la matière première et/ou des composants en produits finis et/ou composants. Un tel système est qualifié de cyclique lorsqu'il fabrique indéfiniment la même séquence de produits. L'ordonnancement d'un système de production cyclique est défini dès que l'on connaît l'instant de début de chaque opération sur la ressource correspondante. Il a été démontré que, quel que soit l'ordonnancement admissible choisi, il est toujours possible d'utiliser la ressource goulot sans interruption. En d'autres termes, il est toujours possible de maximiser la productivité d'un tel système. En conséquence, notre objectif est de trouver l'ordonnancement qui permet d'atteindre la productivité maximale avec un en-cours aussi faible que possible. Nous proposons une heuristique basée sur les réseaux de Petri pour obtenir une solution proche de l'optimum. Nous fournissons également une condition suffisante d'optimalité.

[1] INRIA-Lorraine, 4 rue Marconi, 57070 Metz, FRANCE
[2] Institute for Systems Research, University of Maryland, College Park, MD 20742, USA
[3] INRIA-Lorraine, 4 rue Marconi, 57070 Metz, France and Institute for Systems Research, University of Maryland, College Park, MD 20742, USA

TOWARDS OPTIMIZATION OF CYCLIC PRODUCTION SYSTEMS

FABRICE CHAUVET[1], JEFFREY W. HERRMANN[2] and
JEAN-MARIE PROTH[3]

ABSTRACT

In this paper, the expression "production systems" refers to flow-shops, job-shops, assembly systems, Kanban systems and, in general, to any Discrete Event System (DES) which transforms raw material and/or components into products and/or components. Such a system is said to be cyclic if it provides indefinitely the same sequence of products. A schedule of a cyclic production system is defined as soon as the starting time of each operation on the related resource is known. It has been showed that, whatever the feasible schedule applied to the cyclic production system, it is always possible to fully utilize the bottleneck resource. In other words, it is always possible to maximize the productivity of such a system. As a consequence, we aim at finding the schedule which permits to maximize the productivity with a Work-In-Process (WIP) as small as possible. We propose a heuristic approach based on Petri nets to find a near-optimal, if not optimal, solution. We also give a sufficient condition for a solution to be optimal.

KEY WORDS: Petri nets, scheduling, cyclic production systems, event graphs, WIP

1. INTRODUCTION

A cyclic production system manufactures a set of products at a constant frequency. That is, each resource repeatedly performs a certain set of tasks, and the system produces a certain set of items each cycle. Careful scheduling is required to coordinate the resources so that the system produces items at the highest possible frequency and with the minimal amount of work-in-process inventory (WIP).

EXAMPLE.

This paper considers the scheduling of cyclic production systems and describes a new approach that uses Petri nets. The problem is to determine the time at which each task should begin during the cycle. We assume that the set of tasks does not change, the tasks are non-preemptive, the task processing times are deterministic, and the resources are always available.

Section 2 discusses Petri nets and event graphs. In Section 3, we show that it is always possible to fully utilize the bottleneck machine and achieve maximum throughput, although the WIP depends upon the particular schedule. A sufficient condition for a feasible solution to be optimal is proposed in Section 4. Section 5 presents a powerful heuristic algorithm. This algorithm generates a near-optimal solution. Section 6 concludes the paper.

[1] INRIA-Lorraine, 4 rue Marconi, 57070 Metz, FRANCE
[2] Institute for Systems Research, University of Maryland, College Park, MD 20742, USA
[3] INRIA-Lorraine, 4 rue Marconi, 57070 Metz, France and Institute for Systems Research, University of Maryland, College Park, MD 20742, USA

2. PETRI NETS AND EVENT GRAPHS

Petri nets are a powerful modeling tool for discrete event systems. Event graphs are a special type of Petri net.

2.1. Petri nets

We restrict ourselves to the elementary Petri nets, also called black-and-white nets.

A Petri net PN is a 5-tuple (P, T, A, W, M_0) with the following components:

$P = \{ p_1, p_2, ..., p_q \}$ is the set of places,

$T = \{ t_1, t_2, ..., t_n \}$ is the set of transitions,

$A \subseteq (P \times T) \cup (T \times P)$ is a set of arcs,

$W: A \rightarrow \{ 1, 2, ... \}$ is a weight function,

$M_0: P \rightarrow \{ 1, 2, ... \}$ is the initial marking.

In this paper, transitions represent operations, places represent either buffers for products or storage facilities for information, arcs model the flows of products or information, and tokens represent products or pieces of information.

Places are represented by circles, transitions by bars, and each place contains a number of tokens (represented by dots) equal to its marking.

In Figure 1, we present a Petri net with marking $M_0 = < 2,1,4,2,0 >$, where the i-th component of M_0 is the number of tokens in place p_i.

When no weight is mentioned, the arc has a weight of 1.

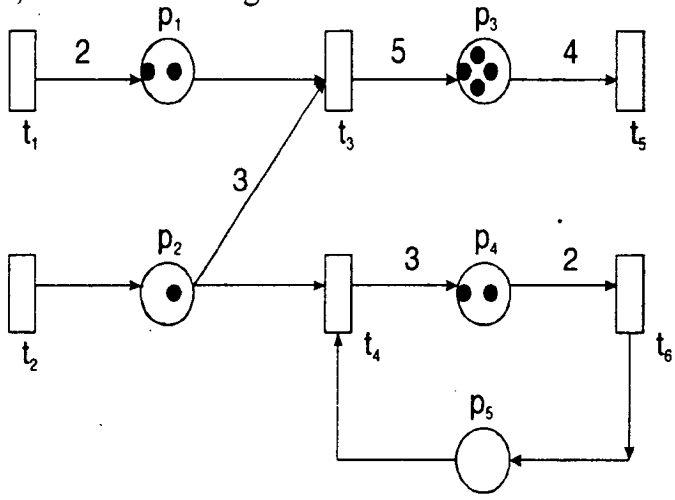


Figure 1: A Petri net

Given a marking M, a transition is said to be enabled if and only if $M(p) \geq W(p,t)$ for all $p \in {}^{\circ}t$., where ${}^{\circ}t$ represents the set of input places of transition t.

A transition with no input places is called a source transition. A source transition is always enabled.

For instance, t_5 is enabled since p_3 contains four tokens, which is equal to the weight of arc (p_3, t_5), while t_3 is not enabled since p_2 contains only one token, which is less than the weight of arc (p_2, t_3). Transitions t_1 and t_2 are both enabled since they are source transitions.

Firing a transition consists of two parts:

(i) removing W(p,t) tokens from each $p \in {}^{\circ}t$, and

(ii) adding W(t,p) tokens to each $p \in t^{\circ}$, where t° is the set of output places of t.

For instance, the marking of the net in Figure 1 becomes $M = < 2,1,4,0,1 >$ after firing t_6. Firing a source transition t adds W(t,p) tokens to each $p \in t^{\circ}$. For instance, firing t_1 adds two tokens to place p_1 while firing t_2 adds one token to p_2. A transition with no output place is called a sink transition. Firing a sink transition t removes W(p,t) tokens from each $p \in {}^{\circ}t$.

An enabled transition may, or may not, be fired.

A timed Petri net is a Petri net for which duration, or firing time (which is deterministic or stochastic), is associated to each transition. In this paper, only deterministic firing times are considered.

The duration associated with a transition represents the time between the instant the tokens disappear from the input places and the instant the tokens appear in the output places. Usually, we consider that tokens continue to belong to the input places of a transition t until the firing of t ends. From a production point of view, this duration represents the time required to perform the operation.

The incidence matrix of a Petri net, say $U = [u_{ij}]$, $i = 1,2,\ldots,q$, $j = 1,2,\ldots,n$, is defined as follows:

$$u_{ij} = \begin{cases} W(t_j, p_i) \text{ if } t_j \in {}^{\circ}p_i \\ -W(p_i, t_j) \text{ if } t_j \in p_i{}^{\circ} \\ 0 \text{ otherwise} \end{cases}$$

where ${}^{\circ}p$ (resp. p°) is the set of input (output) transitions of p.

Note that the incidence matrix reflects the structure of the Petri net provided that the net does not contain self-loops (i.e. loops composed by only one place and one transition).

Let M_0 be an initial marking and σ a sequence of transitions fired starting from M_0. We denote by M the marking obtained after firing the last transition of σ. We define the firing count vector V_σ related to the sequence σ as:

$V_\sigma = (v_1, v_2, \ldots, v_n)$

where v_i ($i = 1,2,\ldots,n$) is the number of times transition t_i appears in σ, and n is the total number of transitions.

The state equation is as follows:

$$M^t = M_0^t + U.V_\sigma^t \qquad (1)$$

where A^t denotes the transpose of A.

Note that if a sequence s of transitions satisfies (1), it is not guaranteed that s is feasible (i.e. that it is possible to fire the sequence of transitions s).

A vector Z is a **p-invariant** if:

(i) $ZU = 0$, (2)

(ii) Z is a q-vector whose components are non-negative integers, and

(iii) at least one of the components of Z is strictly positive.

Let $R(M_0)$ be the set of markings reachable from M_0. One can easily show that, if Z is a p-invariant, then for any M in $R(M_0)$,

$ZM^t = ZM_0^t$ (3)

The proof is made by left-multiplying both sides of (1) by Z, and by using (2). Thus, ZM^t is an invariant, i.e. a linear combination of the place markings that remains constant for any sequence of transition firings.

A vector H is a t-invariant if:

(i) $UH^t = 0$, (4)

(ii) H is a n-vector whose components are non-negative integers,

(iii) at least one of the components of H is strictly positive.

Let σ be a firing sequence and V_σ the related firing count vector. If V_σ is a t-invariant, M_0 the initial marking and M the marking obtained after firing the sequence σ of transitions, then $M = M_0$

2.2. Event graphs (or marked graphs)

An event graph is a Petri net such that each place has exactly one input and one output transition. Furthermore, the weight associated with each arc is 1. Such a Petri net is represented in Figure 2. Note that event graphs may contain elementary circuits. For example, the event graph of Figure 2 contains three elementary circuits: namely $\gamma_1 = <p_1, t_1, p_2, t_3, p_3, t_2>$, $\gamma_2 = <p_4, t_5, p_7, t_4, p_5, t_3>$ and $\gamma_3 = <p_6, t_5, p_7, t_4>$.

It is easy to prove the following:

(i) If we assign one token to each place belonging to one of the elementary circuits and 0 to the other places, then the vector whose elements are these values is a p-invariant. More precisely, this vector is a minimal p-invariant, i.e. a p-invariant that contains no other p-invariant. That is, the number of tokens is any elementary circuit is invariant by any transition firing. We owe this result to COMMONER *et al.* [2].

(ii) The n-vector whose components are all equal to 1 is a t-invariant. That is, the marking returns to the initial marking after firing each transition exactly once.

The following result is also due to COMMONER *et al.* [2].

Result 1: A strongly connected event graph is guaranteed to be deadlock-free if and only if every elementary circuit contains at least one token.

In the example given in Figure 2, the token in p_2 concerns γ_1 while the token in p_7 concerns γ_2 and γ_3. Thus, this event graph is deadlock-free.

We assume that the times associated to the transitions, i.e. the manufacturing times, are deterministic. For any elementary circuit γ, we define the cycle time $C(\gamma)$ as follows:

$C(\gamma) = \mu(\gamma) / M(\gamma)$ (5)

where $\mu(\gamma)$ is the sum of the firing times of the transitions belonging to γ, and $M(\gamma)$ is the number of tokens circulating in γ.

If M_o is the initial marking, we know that $M(\gamma) = M_o(\gamma)$, for all M in $R(M_o)$. Thus $C(\gamma)$ is an invariant. Let C* be the maximal cycle time among all elementary circuits:

$C^* = \max \{C(\gamma): \gamma \text{ in } \Gamma\}$, (6)

where Γ is the set of elementary circuits of the strongly connected event graph. Any γ in Γ such that $C(\gamma) = C^*$ is called a critical circuit.

In order to guarantee that only a single firing of a certain transition may occur at time t, we introduce a self-loop to each transition with initially one token in each self-loop place.

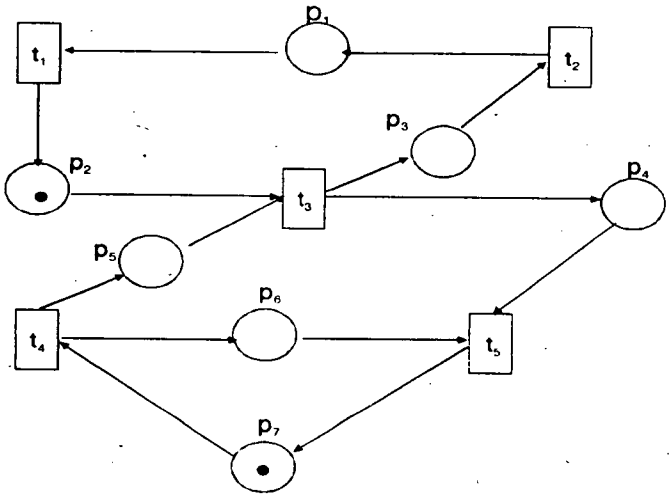


Figure 2: An event graph

Consider the event graph of Figure 2. Suppose the marking is < 2, 1, 4, 0, 3, 2, 1 >, there is a self-loop associated with each transition, and the place of this loop contains one token (the self-loops are not represented in Figure 2), and the times associated with t_1, t_2, t_3, t_4 and t_5 are 5, 3, 1, 2, and 6 respectively.

In this case, $C(\gamma_1)=9/7$, $C(\gamma_2)=9/4=3/2$ and $C(\gamma_3)=8/3$. But the self-loops are also elementary circuits. Thus their cycle times should also be taken into account. The greatest cycle time among the self-loops is 6, which is the cycle time of the self-loop corresponding to t5. Since this value is greater than $C(\gamma_1)=9/7$, $C(\gamma_2)=9/4=3/2$ and $C(\gamma_3)=8/3$, the maximal cycle time $C^*=6$.

Consider the firing policy called Earliest Operating Mode (EOM), where transitions fire as soon as they are enabled. Under this policy, the operation of the system becomes periodic after a finite time. Periodicity means that there exist two integers n_0 and K such that:

$S_t(n + K) = S_t(n) + KC^*$ for all $n \geq n_0$ and all t in T.

where $S_t(k)$ is the starting time of the k-th firing of transition t and K is the period.

The following result is due to CHRETIENNE [4].

Result 2: Provided that the event graph is strongly connected and that the EOM policy applies, the cycle time of the model is C^* in a steady state. In other words, the firing rate of all transitions in steady state is $\lambda = 1 / C^*$.

This result provides the productivity of a manufacturing system that is modeled by a strongly connected event graph, assuming that the WIP is known and that EOM applies.

Several algorithms have been proposed to reach a given cycle time while minimizing a linear combination of the markings, the coefficients of which are the components of a p-invariant. They can be found in LAFTIT et al. [5]. In particular, an adjustment heuristic algorithm, which can be used in practice for models of any size, is proposed in [5]. But, in this algorithm, the schedule is

given. Our goal is more ambitious since we are looking for the schedule that yields the maximal productivity (or throughput) with minimal WIP.

3. MAXIMIZING PRODUCTIVITY FOR A GIVEN SCHEDULE

In this section, we consider the objective of maximizing the productivity. This is equivalent to reducing the length of the cycle and utilizing the bottleneck machine fully. Two approaches are proposed.

The first one, called the elementary approach, assigns the tasks to the resources in an arbitrary order, without taking into account the WIP, and fixing the cycle time to the time required by the bottleneck machine to complete a cycle if it is fully utilized. The WIP is adjusted to reach this goal.

Any cyclic production system can be modeled as an event graph when the schedule is given a priori, as showed in [6], [7] and [8]. It is then possible to derive the minimal WIP using the properties of the event graphs. We refer to this approach as the event graph approach.

Before reminding the reader of these methods, we introduce a simple example, which will be used in this paper to illustrate the proposed approaches.

3.1. Notation

We start by introducing some additional notation that precisely formulates the problem.

Let b = 1, ..., n be the index over the different products. (The cyclic production system produces one unit of each product each cycle.)

Let i = 1, ..., p be the index over all of the different tasks that need to be performed during each cycle. Let r(i) be the resource that performs task i.

Let d(i) be the duration of task i.

Let S(b) be the sequence of tasks required to produce one unit of product b. Let q(b) be the number of tasks required to produce one unit of product b. (We assume that there is a simple sequence of tasks. One could easily extend our results to the case where some tasks can be done in parallel.)

Let E(b) be the set of immediate precedence constraints. If (i,j) is in E(b), then task j follows task i in S(b). Then, i = succ(j), and j = pred(i).

Let r = 1, ..., R be the index over the resources.

x(i,r) = 1 if r(i) = r and 0 otherwise.

Then, let C(r) be the total processing requirements on resource r:

$C(r) = x(1,r)d(1) + ... + x(p,r)d(p)$.

Let C* be the maximum processing requirements: $C^* = \max \{C(1), ..., C(R)\}$.

3.2. An Example

For the sake of clarity and simplicity, we consider a job shop that has R = 4 machines and produces n = 3 products. There are p = 11 different tasks that need to be performed. The following table lists the tasks (in sequence for each product), the resources that perform the tasks, and the task durations.

Product	Task	Resource	Duration
1	1	1	3
	2	2	1
	3	3	5
	4	4	2
2	5	4	8
	6	3	7
	7	1	5
	8	2	9
3	9	3	5
	10	4	2
	11	2	3

We then derive the total processing requirements on each resource (machine) during a cycle: $C(1) = d(1) + d(7) = 3 + 5 = 8$. Similarly, $C(2) = 13$, $C(3) = 17$, and $C(4) = 12$. $C^* = C(3) = 17$, so the bottleneck machine is machine 3, and the minimal cycle time is 17 time units. Thus, the maximal productivity (throughput) is 3/17 items per time unit.

3.2. The Elementary Approach

We begin by presenting the algorithm. Afterwards, we will apply it to the example presented above. This algorithm determines a feasible schedule and how many cycles are needed to complete one item of each product, given that schedule.

Algorithm 1.

1. For each resource, schedule the tasks that require that resource in any order. Let b(i) and c(i) denote the start and end times of task i. All tasks must start in the interval [0, C*], and the resource can perform at most one task at a time. (Thus, if a task ends at time $t > C^*$, the resource can perform no other task during the interval [0, t-C*].) No tasks can be preempted.
2. For each product b = 1, ..., n, perform Step 3 for the first task in S(b) and perform Step 4 for each remaining task.
3. Let i be the first task in S(b). Assign the label w(i) = 0 to this task.
4. Let i be the next unlabeled task in S(b) and let j = pred(i). Label task i as follows: If $c(j) > b(i)$, then let $w(i) = w(j) + 1$. Otherwise, let $w(i) = w(j)$.
5. For each product b = 1, ..., n, let g be the first task in S(b) and h be the last task in S(b). Then, calculate the product cycle time as follows:

$$T(b) = (w(h) - w(g))C^* + c(h) - b(g).$$

Since w(g) = 0 by definition,

$$T(b) = w(h)C^* + c(h) - b(g).$$

6. The total average WIP is $(T(1) + ... + T(n) / C^*$ items.

Let us apply this algorithm to the example. Recall that $C^* = 17$. In Step 1, we create the following schedule arbitrarily: Machine 1 performs task 1 and then task 7. Machine 2 performs

task 8, task 2, and then task 11. Machine 3 performs task 3, task 6, and then task 9. Machine 4 performs task 10, task 4, and finally task 5.

The following table lists the start and end times of each task on each machine.

Machine	Task	Start	End
1	1	0	3
	7	3	8
2	8	0	9
	2	9	10
	11	10	13
3	3	0	5
	6	5	12
	9	12	17
4	10	0	2
	4	2	4
	5	4	12

Now, we repeat Steps 2, 3, and 4 and label each task. Consider product 1. The production system must perform tasks 1, 2, 3, and 4 to complete one item of product 1. Given the above schedule, since task 2 starts after task 1 ends, a particular item can undergo both tasks in the same cycle (which we call cycle 0). However, since task 3 starts before task 2 ends, the item must undergo task 3 in the next cycle (cycle 1). Likewise, since task 4 starts before task 3 ends, the item must undergo task 4 in the cycle after that (cycle 2). The total cycle time for that item T(1) = 2*17 + 4 - 0 = 38 time units. Likewise, T(2) = 56, and T(3) = 18. The following table lists the labels for each task.

Product	Task	Label
1	1	0
	2	0
	3	1
	4	2
2	5	0
	6	1
	7	2
	8	3
3	9	0
	10	1
	11	1

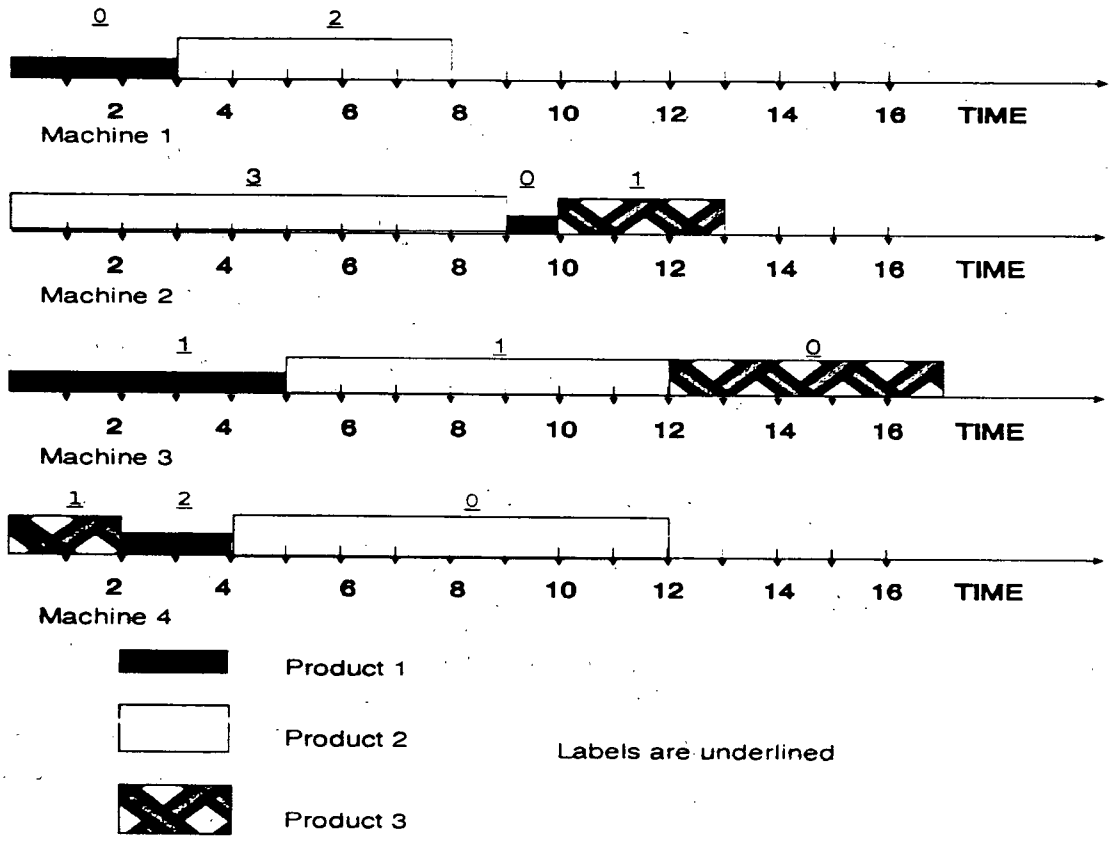


Fig. 3: The elementary approach applied to the illustrative example

The throughput for each product is one item per 17 time units. Since WIP equals cycle time times throughput (by Little's Law), there are, on average, 38/17 items of product 1 in the system, 56/17 items of product 2, and 18/17 items of product 3. The total average WIP is thus (38+56+18)/17 = 6.59 items

3.3. The Event Graph Approach

In this approach, the schedule is given a priori. The algorithm is based on an event graph model of the cyclic manufacturing system. In this subsection, we show how to model a cyclic production system using event graphs, and we use an adjustment algorithm to generate a near optimal solution to the problem.

3.3.1. Modeling the production system.

Create the event graph model of a cyclic production system as follows:

(i) Model the manufacturing process of each product. Each transition represents a task and each place a buffer. Each transition firing corresponds to the execution of a task and the firing time of the transition is the time required to perform this task. Each manufacturing process is reproduced as many times as necessary to yield the given production ratios. For instance, in the example introduced above, every manufacturing process is represented once, since one item of each product is completed during each cycle.

(ii) Model the cyclic operation mode of the system by assuming that a new item is available for processing as soon as an item of the same product is completed. This is modeled by adding an output place to the last transition of each manufacturing process model and by connecting this place to the first transition of the same model. Thus, we obtain elementary circuits called process circuits.

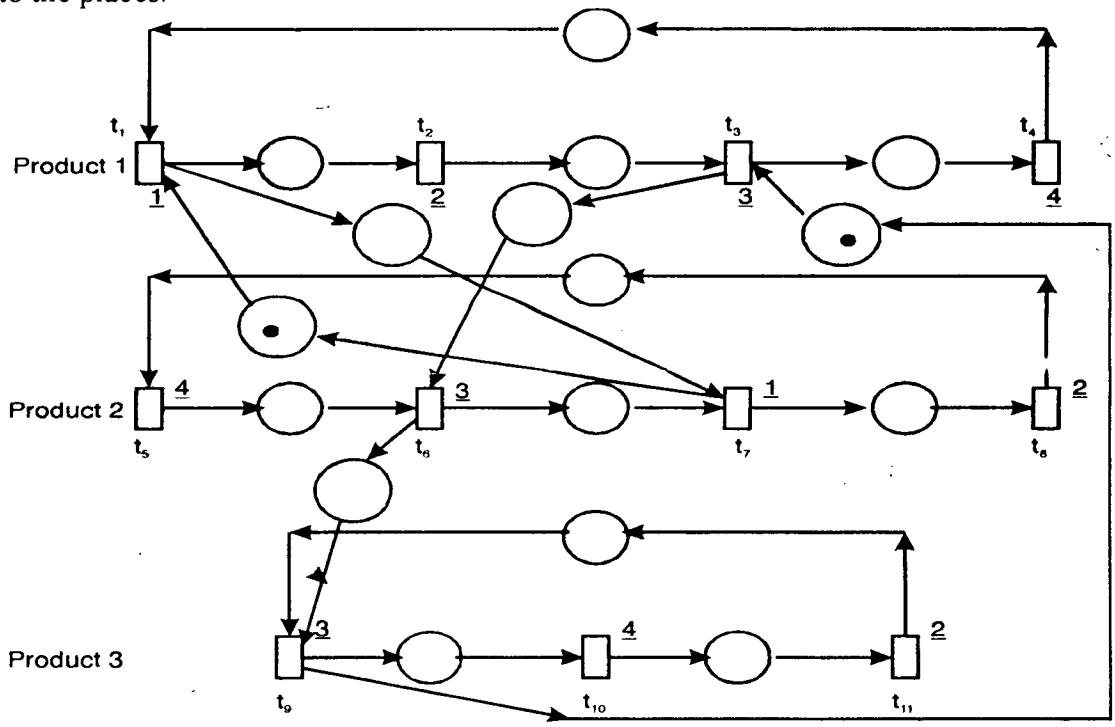
(iii) The final step is to model the sequence of tasks on each resource. This is done by connecting in a unique circuit all of the transitions that correspond to tasks performed by the same resource. The order of transitions in these new elementary circuits (called command circuits) is determined by the task sequence.

Tokens circulating in the process circuits represent products whereas tokens in the command circuits represent resource status information. Note that there is exactly one token in each command circuit, since a resource can perform at most one task at a time, and an elementary circuit without a token blocks after a finite number of transition firings.

Mixed circuits are made with parts of command circuits and parts of process circuits.

The number of command circuits is equal to the number of resources, while the number of process circuits is equal to the number of items produced during each cycle. The number of mixed circuits depends on the number of resources, the number of products, and the schedule.

In Figure 4, we present the event graph model of the production system introduced in Subsection 3.1. For the sake of clarity, we do not represent the command circuits corresponding to machines 2 and 4. Since a place can always be defined by its input and output transitions, we do not give names to the places.



The machine numbers are underlined

The operation numbers are the indexes of the transitions

Fig. 4: Event graph model

In this model, $<t_1, t_2, t_3, t_4>$ is a process circuit, $<t_3, t_6, t_9>$ is a command circuit, and $<t_3, t_6, t_7, t_1, t_2>$ is a mixed circuit.

3.3.2. The adjustment algorithm

Since this algorithm has been already presented in [5], we restrict ourselves to a brief presentation of it. We know that only one token should be put in each command circuit. For a given sequence of tasks, we know that this token should be in the input place of the transition corresponding to the first task. Such a token represents information. To maximize productivity (throughput), we need enough tokens in the process circuits so that the longest circuit is a command circuit. In addition, we want to reach this goal with as few tokens as possible, since each token introduced in a process circuit represents an item, and we want to fully utilize the bottleneck machine while minimizing the WIP.

This problem can be modeled as a 0-1 integer-programming problem. Unfortunately, the size of the problem increases proportionally to the number of elementary circuits, and the number of mixed circuits increases exponentially with the number of transitions. Thus, heuristic algorithms like the adjustment algorithm are necessary. The adjustment algorithm starts with a feasible solution and then proceeds iteratively by removing tokens until this would create an infeasible solution. It is based on the following result whose proof can be found in [5] and [7]. In this paper, we restrict the formulation of the result to our specific need.

Result 1:

For a place p, a marking M, and a cycle time C*, let $w(M, p, C^*) = \min \{ M(\gamma) - \mu(\gamma)/C^* : \gamma \text{ in } \Gamma p \}$, where $M(\gamma)$ is the total number of tokens in the places belonging to circuit γ, $\mu(\gamma)$ is the total duration of the transitions in circuit γ, and Γp is the set of elementary circuits containing p.

If $\mathbf{M}(C^*)$ is the set of markings M such that $0 \leq w(M, p, C^*) \leq 1$ for all places p, then any M in $\mathbf{M}(C^*)$ is feasible (the system can achieve the maximal productivity). Furthermore, if M is in $\mathbf{M}(C^*)$, $M_1 \leq M$, and $M_1 \neq M$ (i.e. $M_1(p) \leq M(p)$ for all places p, and there exists at least one place p_1 such that $M_1(p_1) < M(p_1)$), then M_1 is not feasible.

The adjustment algorithm can be summarized as follows for the particular case under consideration:

Algorithm 2

1. Generate a feasible marking M by placing one token in each place of the process circuits in addition to the tokens which are in the command circuits.

2. If $w(M, p, C^*) < 1$ for all places, then stop the computation and keep M as the solution of the problem. The number of tokens in the process circuits equals the average WIP in the system.

3. Otherwise, let P1 be the set of places p such that $w(M, p, C^*) \geq 1$. For each p in P1, construct the following two markings: If $M(p) > 0$, let $A_p = M$; otherwise, let A_p be a marking in R(M) such that $A_p(p) > 0$. Then, construct B_p as follows: $B_p(q) = A_p(q)$ for all $q \neq p$; $B_p(p) = A_p(p) - 1$. Now, define the function f(p) as follows:

$f(p) = \sum_{q \in P} [w(M, q, C^*) - w(B_p, q, C^*)]$

Choose the p^* in P1 that minimizes f(p).

4. Replace M by the marking B_{p^*}.

5. Go to Step 2.

3.3. Comparing the Event Graph and Elementary Approaches

Both methods require a given sequence of tasks for each resource. However, they compute the WIP requirements differently.

In the elementary approach, the start times of the tasks are calculated based on the given sequences, and then the WIP is calculated. The event graph method determines the start times based on the WIP (the tokens), which allows it to find better task start times. Consider for instance Figure 3. If we postpone task 7 and start it at time 12 (instead of at time 3), we reduce the number of cycles required to complete one item of product 2, and we reduce the WIP. (We may be able to achieve the optimal solution by solving the integer-programming problem when the size of the model is small enough.)

In this sense, we can claim that the event graph approach is preferable to the elementary approach, but it requires more computation.

4. SUFFICIENT CONDITION FOR OPTIMALITY

Definition

Let b = 1, ..., n be the set of products that are manufactured during each cycle, and let T(b) be the total time necessary to complete one item of product b according to the cyclic schedule under consideration. In other words, T(b) is the item cycle time. Then the WIP Q is defined as $(T(1) + ... + T(n))/C^*$, where C^* is the duration of one cycle.

Result 2

Let b = 1, 2, ..., n be the set of products that are manufactured during each cycle, let C^* be the duration of the cycle, let T(b) be the item cycle times, and let P(b) be the sum of the durations of the tasks required to produce one item of product b. If, for a given schedule, $T(b)/C^* \leq \lceil P(b)/C^* \rceil$ for all products b, then the schedule is optimal, in the sense that the maximum productivity is obtained with the minimum WIP. As usual, $\lceil a \rceil$ is the smallest integer greater than or equal to a.

Proof

$\lceil T(b)/C^* \rceil$ is the minimal number of cycle times required to complete a product of type i. Thus, the best schedule possible to complete a product of type i during each cycle is a schedule for which $\lceil T(b)/C^* \rceil$ products of type i, at different stage of completion, can be observed in the system during each cycle. In other words, the part of the WIP corresponding to i is less than or equal to $\lceil T(b)/C^* \rceil$. Q.E.D.

This result provides the best possible WIP. It permits to decide if a solution is optimal and, otherwise, to evaluate the maximal "distance" to the optimal solution.

5. A NEW APPROACH

This section presents an approach that extends the event graph approach presented previously. Called the Construction Approach, it requires only a sequence of tasks for the bottleneck resource and creates sequences for the other resources.

5.1. Introduction of the Construction Approach

The event graph approach presented above requires, for each resource, the sequence of tasks to be performed by each resource. The construction approach requires a sequence of tasks for the bottleneck resource (or one of the bottleneck resources) only. Sequences for the other resources are constructed by the algorithm.

The idea behind this algorithm is quite simple: since we want the bottleneck resource to be fully utilized, the token assigned to that resource's command circuit should never wait in a place. As a consequence, as soon as a transition of this command circuit is fired, the next transition should be enabled, and its firing must start immediately. Because each transition has two input places (one belonging to the command circuit and one to a process circuit), the input place belonging to the process circuit should contain at least one token at the time the transition must fire. We introduce this token to this place when it is needed. When a transition firing ends, one token appears in the output place belonging to the process circuit and another token appears in the output place belonging to the command circuit. We continue to fire the transitions in the command circuit under consideration until one cycle is completed. This completes the first step of the algorithm.

The second step of the algorithm simultaneously builds the command circuits of the other resources (i.e. the remainder of the schedule) and determines the start times of the other tasks.

5.2 The Construction Algorithm

This section presents the algorithm. We will give some examples in the next section.

We assume that transition t_i corresponds to task i. Recall that r(i) is the resource that performs task i.

Algorithm 3

Given: A sequence of tasks for a bottleneck resource.

Initialization: Construct all of the process circuits. The values z(i,j) for all places (i,j) in the process circuits are undefined. For each transition t_j, construct a second input place u_j. These control places will form the command circuits. The values $w(u_j)$ for all control places are undefined. C* equals the total processing time on the bottleneck resource.

Part A: Fire the transitions of the command circuit of a bottleneck resource.

1. Construct a command circuit corresponding to the sequence of tasks that bottleneck resource performs. If the resource performs tasks $i_1, i_2, ..., i_n$ during each cycle, let $< t_{i_1}, t_{i_2}, ..., t_{i_n} >$ be the command circuit. Hereafter, let k+1 = 1 if k = n and k-1 = n if k = 1. For k = 1, ..., n, connect each transition t_{i_k} to the input place $u_{i_{k+1}}$. Introduce a token in place u_{i_1} and set $w(u_{i_1}) = 0$.
2. For k = 1, 2, ..., n, do the following steps:
   2.1. Let b be the product that requires task i_k. Let t_j be the transition that immediately precedes t_{i_k} in the process circuit for product b. Add a token to place (t_j, t_{i_k}) and set $z(j, i_k) = z(i_{k-1}, i_k)$.

2.2. Fire transition t_{i_k}. Set $z^*(i_k) = w(u_{i_k}) + \mu(i_k)$, where $\mu(i_k)$ is the duration of t_{i_k}. Remove one token from places u_{i_k} and (t_j, t_{i_k}).

2.3. Let t_h be the transition that immediately follows t_{i_k} in the process circuit for product b.

2.4. Add one token to places u_{i_k+1} and (t_{i_k}, t_h). Set $w(u_{i_k+1}) = z^*(i_k)$. Set $z(i_k, h) = z^*(i_k)$.

3. Let PN be the resulting event graph. Let Z be the set of values applied to places in the process circuits. Let W be the set of values applied to the control places.

Part B. Perform a cycle while building the missing command circuits.

4. If there are any process circuits with no tokens, then, for each such circuit, add one token to the place (i,j) that is the input place for the first transition in that process circuit, and set $z(i,j) = 0$.

   Mark with the label "NB" all of the places that belong to a process circuit and contain a token, where "NB" stands for "not blocked".

5. Let E be the set of places that belong to a process circuit, contain a token, and are labeled NB. If E is empty, then go to Step 9. Otherwise, select the place (i, j) in E with the minimal z(i,j).

6. Consider transition t_j, the resource r(j) that performs task j, and the other tasks that resource r(j) performs. Let F be the set of corresponding transitions.

   If no transitions in F have been fired at this point, t_j is the first transition of the command circuit for r(j). Add a token to u_j. If $w(u_j)$ is undefined, set $w(u_j) = z(i,j)$.

   Otherwise, if t_k is the transition in F that was fired last, t_j follows t_k in the command circuit for r(j). Connect transition t_k to place u_j. Add a token to u_j. Let $w(u_j) = z^*(k)$.

   In either case, let $z^*(j) = \max \{z(i,j), w(u_j)\} + \mu(j)$.

   If t_j is the only unfired transition in F, then let t_m be the transition (in this command circuit) that was fired first. Connect transition t_j to the place u_m.

   Let t_s be the transition that follows t_j in the process circuit.

7. If z(j,s) is undefined, fire t_j. Remove one token from the place (i,j) and remove one token from the place u_j. Add one token to the place (j, s). Define $z(j,s) = z^*(j)$. Remove the label NB from (i,j) and apply the label NB to (j,s). Go to Step 8.3.

8. Otherwise, z(j,s) is defined.

   8.1. If $z^*(j) \leq z(j,s) + C^*$, then fire t_j. Remove one token from the place (i,j) and remove one token from the place u_j. Add one token to the place (j, s). Redefine $z(j,s) = z^*(j)$. Remove the label NB from (i,j) and apply the label B to (j,s). This means that the new token in place (j,s) is blocked and cannot enable transition t_s. Go to Step 8.3.

   8.2 Otherwise, $z^*(j) > z(j,s) + C^*$. The solution is infeasible. Reset the event graph to PN. Update M by adding a token to place (i,j). Reset the values on the process circuit places to the set Z. Update Z by defining $z(i,j) = z^*(j) - \mu(j) - C^*$. Reset the values on the control places to the set W. Return to Step 4.

   8.3 If t_j was the last unfired transition in F, then let t_m be the transition (in this command circuit) that was fired first. If $z^*(j) > w(u_m) + C^*$, then the PN was

infeasible. Reset the event graph to PN. Reset the values on the process circuit places to the set Z. Reset the values on the control places to the set W. Update W by defining $w(u_m) = z^*(j) - C^*$. Return to Step 4.

9. Create a feasible schedule as follows: Each task j ends at time $z^*(j)$. The task begins at time $z^*(j) - \mu(j)$. (Any task that starts outside [0, C*] can be shifted to this interval by adding or subtracting a multiple of C*.) Use the elementary approach (Algorithm 1) to calculate the cycle time and WIP of the schedule.

5.3 Example.

Consider, for instance, the example given in Figure 4. The bottleneck is Machine 3, and we are given the task sequence (3, 6, 9) for this resource.

Task 3 is the first task on Machine 3. Transition t_3 has two input places: place (2,3), which is in the process circuit of Product 1, and place (9,3), which is in the command circuit of Machine 3. (This place is also denoted u_3.)

To enable transition t_3, there must be at time 0 a token in both places. We introduce a token in each place and label the places with time 0. $w(u_3) = 0$. $z(2,3) = 0$. Because the firing time of t_3 is 5, a token appears at time $z^*(3) = 5$ in place (t_3, t_6), which belongs to the command circuit, and in place (t_3, t_4), which belongs to the process circuit of Product 1. $w(u_6) = 5$. $z(3,4) = 5$.

For transition t_6 to fire immediately at time 5, there must be, at that time, a token in place (5,6). Add a token there, and set $z(5,6) = 5$. Because the firing time of t_6 equals 7, a token appears at time $z^*(6) = 12$ in place (6,9) and in place (6,7). $w(u_9) = 12$. $z(6,7) = 12$.

The last transition to fire is t_9. Since it should fire at time 12, there must be, at that time, a token in place (11,9). Add a token there, and set $z(11,9) = 12$. Firing t_9 adds, at time $z^*(9) = 17$, one token to place (t_9, t_{10}) and one token to place (t_9, t_3). $w(u_3) = 17$. $z(9,10) = 17$.

At the end of Part A, the state of the event graph can be summarized by Table 1, which lists only the places belonging to the process circuits.

Table 1: State of the system at the end of the first step

Places (i,j)	z(i,j)	Number of tokens
(1, 2)	-	0
(2, 3)	0	0
(3, 4)	5	1
(4, 1)	-	0
(5, 6)	5	0
(6, 7)	12	1
(7, 8)	-	0
(8, 5)	-	0
(9, 10)	17	1
(10, 11)	-	0
(11, 9)	12	0

The second step of the algorithm builds the command circuits of the other machines and determines the start times of the remaining tasks. We start by selecting the place (i,j) that belongs

to a process circuit, contains a token, and has the smallest value z(i,j). In this example, that is place (3,4).

Transition t_4 will be part of the command circuit for Machine 4, which performs task (i.e., r(4) = 4). The transitions in this circuit are t_4, t_5, and t_{10}. None have been fired, so t_4 is the first transition in the circuit, and it can fire as soon as possible. So, add a token to u_4, and set $w(u_4) = z(3,4) = 5$. Fire t_4, which ends at $z^*(4) = 7$. Remove the tokens in (3,4) and u_4, and add a token to (4,1), the next place in the process circuit. Set z(4,1) = 7.

Repeat the above steps for t1 and t2. $w(u_1) = 7$. $z^*(1) = 10$. $z(1,2) = 10$. $w(u_2) = 10$. $z^*(2) = 11$. Because $z^*(2) < z(2,3) + C^* = 17$, redefine z(2,3) = 11. The token that enters place (2,3) becomes blocked because t_3 has fired previously.

Now consider place (6,7), with z(6,7) = 12. The other transition in the command circuit for Machine 1 (r(7) = 1) is t_1. Thus, connect t_1 to u_7, add a token to u_7, and let $w(u_7) = z^*(1) = 10$. This represents the fact that Machine 1 is available at time 10. Fire t_7, which ends at $z^*(7) = 12+5 = 17$. Remove the tokens in (6,7) and u_7, and add a token to (7,8). Set z(7,8) = 17. Note $z^*(7) < w(u_1) + C^* = 24$, so this command circuit is feasible.

Similarly, the next transition to fire is t_8. $w(u_8) = z^*(2) = 11$. $z^*(8) = 17+9 = 26$. A token moves to place (8,5), and z(8,5) = 26. Then t_{10} fires. $w(u_{10}) = z^*(4) = 7$. $z^*(10) = 17+2 = 19$. A token moves to place (10,11), and z(10,11) = 19.

Now, consider t_{11}. $w(u_{11}) = z^*(8) = 26$. $z^*(11) = 26+3 = 29$. A token moves to place (11,9), but z(11,9) = 12. Since $z^*(11) = z(11,9) + C^* = 29$, this is feasible, so redefine z(11,9) = 29. However, t_{11} was the last unfired transition in the command circuit for Machine 2. Transition t_2 was the first transition fired. Because $z^*(11) > w(u_2) + C^* = 27$, this command circuit is infeasible. Reset the event graph to the state achieved at the end of Part A. However, now let $w(u_2) = 29-17 = 12$.

Returning to Step 4, transitions t_4, t_1, and t_2 fire. However, this time $z^*(2) = 12+1 = 13$. $z^*(2) < z(2,3) + C^* = 17$, so redefine z(2,3) = 13.

Then, transitions t_8, t_{10}, and t_{11} fire as before. $w(u_8) = 13$, but z(7,8) = 17 is larger, so $z^*(8)$ remains 26. This time, $z^*(11) = w(u_2) + C^* = 29$.

Finally, consider place (8,5). $z(8,5) = z^*(8) = 26$. t_5 is the last unfired transition in the command circuit for Machine 4, so $w(u_5) = z^*(10) = 19$. $z^*(5) = 26+8 = 34$. However, z(5,6) = 5. Because $z^*(5) > z(5,6) + C^* = 22$, the solution is infeasible. Reset the event graph to the state achieved at the end of Part A. Define $z(8,5) = z^*(5) - \mu(5) - C^* = 9$, and add a token to place (8,5). This will allow t_5 to fire at time 9.

Now, with both these changes in place, $w(u_2) = 12$, and z(8,5) = 9, and there is a fourth token in place (8,5).

Returning to Step 4, transitions t_4 and t_1 fire. Now, consider place (4,5). z(4,5) = 9. Because t_4 has already fired, $w(u_5) = z^*(4) = 7$, and $z^*(5) = 9+8 = 17$. Since $z(5) < z(5,6) + C^* = 22$, this is feasible, but the token in (5,6) is blocked.

Next transition t_2 fires. $z^*(2) = 12+1 = 13$. $z^*(2) < z(2,3) + C^* = 17$, so redefine z(2,3) = 13.

Then, transitions t_8, t_{10}, and t_{11} fire as before. $w(u_8) = 13$, but z(7,8) = 17 is larger, so $z^*(8)$ remains 26. Since $z^*(8) = z(8,5) + C^* = 26$, this is feasible, but the token in (8,5) is blocked.

Since t_5 has already fired, $w(u_{10}) = z^*(5) = 17$. $z^*(10)$ remains 19. $z^*(10) < w(u_4) + C^* = 5+17 = 22$, so this command circuit is feasible. Since $z^*(11) = w(u_2) + C^* = 29$, this command circuit is also feasible. And since $z^*(11) = z(11,9) + C^* = 29$, this process circuit is feasible. Thus, we have a feasible solution. Table 2 lists the task start and end times (shifted to the interval [0,17]). The command circuits follow:

Machine 1: <t1, t7, t1>
Machine 2: <t8, t11, t2, t8>
Machine 3: <t3, t6, t9, t3>
Machine 4: <t4, t5, t10, t4>

Table 2.

Machine	Task	Start	End
1	1	7	10
	7	12	17
2	8	0	9
	11	9	12
	2	12	13
3	3	0	5
	6	5	12
	9	12	17
4	10	0	2
	4	5	7
	5	9	17

Given this schedule, the elementary algorithm yields the cycle time for each product. T(1) = 17. T(2) = 34. T(3) = 17. The total average WIP is thus (17+34+17)/17 = 4 items.

6. CONCLUSION

The approach presented in this algorithm is a heuristic approach. It requires the command circuit corresponding to one of the bottleneck resources. The first step of the algorithm consists in placing the operations performed by the selected bottleneck machine according to the order defined by the selected command circuit. The second step is performed iteratively, adding one token in a process circuit at each iteration until the number of tokens is great enough to perform all the operations during one cycle C*. At each iteration of the second step, the algorithm places the operations successively, firing first the transition having one token in its input place belonging to the process circuit for the longest period. We thus try to reduce the production cycle applying local decision. When the algorithm fails to place all the operations during one cycle, the second step restarts after relaxing the WIP in two ways: (i) one token is added, and (ii) this token is added in the place where the event graph reached an unfeasible state in the previous iteration.

As the reader can see, two a priori decisions should be made when launching the algorithm, that is:

(i) choose a command circuit for the bottleneck resource (or for one of the bottleneck resources),

(ii) introduce a rule to decide the order the transitions will be fired.

Using Result 2, we are able to claim that some of the results are optimal. In practice, we have been able to claim that most of the results obtained using Algorithm 3 were optimal. The conclusion remains open when the conditions presented in Result 2 do not apply.

Future researches will focus on these two aspects.

Furthermore, a sufficient condition as been proposed to decide if a schedule is optimal. This condition is based on a realistic hypothesis: if a product is completed during a cycle, it becomes available only at the end of the cycle.

It should be noticed that, to our knowledge, none of the existing heuristics is as flexible as the one proposed in this paper. The algorithm reaches a solution that satisfies the sufficient condition for optimality most of the time for one of the command circuits. The test has been made on a Power Macintosh G3 for systems with up to eight machines and ten products, but the size of these examples could be extended.

Bibliography

[1] Reisig W., "Petri Nets with Individual Tokens", *Informatik-Fachberichte*, vol. 66, no. 21, pp. 229-249, 1983.

[2] Commoner F., Holt A., Even S. and Pnueli A., "Marked Directed Graphs", *Journal of Computer and System Science*, vol. 5, no. 5, pp. 511-523, 1971.

[3] Murata T., "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541-580, April 1989.

[4] Chretienne P., *Les réseaux de Petri temporisés*, Université Paris VI, Paris, France, Thése d'Etat, 1983.

[5] Laftit S., Proth J-M., Xie X-L., "Optimization of Invariant Criteria for Event Graphs, *IEEE Transactions on Automatic Control*, vol. 37, no. 5, pp. 547-555, May 1992.

[6] Hillion H.P., Proth J-M., "Performance Evaluation of Job-Shop Systems Using Timed Event-Graphs, *IEEE Transactions on Automatic Control*, vol. 34, no. 1, pp. 3-9, January 1989.

[7] Proth J-M. and Xie X.L., *Petri Nets. A Tool for Design and Management of Manufacturing Systems*, John Wiley and Sons Ltd, UK, 1996.

[8] Minis .I. and Proth J-M., "Production Management in A Petri Net Environment", *Recherche . Opérationnelle/ Operations Research*, vol. 29, no. 3, pp. 321-352, 1995.

Unité de recherche INRIA Lorraine
Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 Villers lès Nancy Cedex (France)

Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot St Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - B.P. 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, B.P. 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399